Lecture 5: February 23, 2001

Today:

- Priority Queues
- Heaps and Heapsort

Data structures

- maintain data
- support various useful operations

Example: Priority Queues

Support these operations:

- INSERT
- EXTRACT-MIN - Remove and return minimum

Applications:

- sorting: insert all elements, extract all elements (in sorted order)
- shortest paths etc (will see later)

Implementations

Priority queues can have many implementations !

E.g., sorted linked-list:

- INSERT - $\Theta(n)$ time
- EXTRACT-MIN - $O(1)$

Heaps

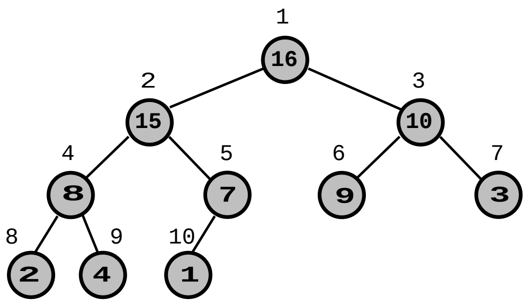
Faster implementation of PQ's using *heaps.*

- *Binary* heap A:
  - Array A with add'l attribute *Heap-size*$[A]$.
  - Viewed as a nearly complete binary tree
  - **heap property:**

    $$A[\text{parent}(x)] \geq A[x]$$

    This defines a *max-heap*
  - There are also min-heaps and k-ary heaps

Heaps: Example



- Notice the implicit tree links:
  Children of node i are $2i$ and $2i + 1$
- Why is this useful?
  Multiplication by 2 is a left shift in binary
  or add to self – **fast**
- Which part of tree is empty?

Heaps: Extract-Max

HEAP-EXTRACT-MAX(A)

$\triangleright$ Removes and returns largest element of A
$max \leftarrow A[1]$
$n \leftarrow Heap\text{-}size[A]$
$A[1] \leftarrow A[n]$
$Heap\text{-}size[A] \leftarrow n - 1$
HEAPIFY($A, 1$) $\triangleright$ Remakes heap
return *max*

Running time? $\Theta(1) +$ HEAPIFY time.

Heaps: Heapify

HEAPIFY(A, i):

- i is index into array A
- Binary trees rooted at LEFT(i) and RIGHT(i) are heaps
- But, $A[i]$ may be smaller than its children, thus violating the heap property.
- HEAPIFY makes A a heap once more.
- How?
  Move $A[i]$ down in heap until heap property
  is satisfied.

Heaps: Heapify

n is total number of elements

HEAPIFY(A, i)

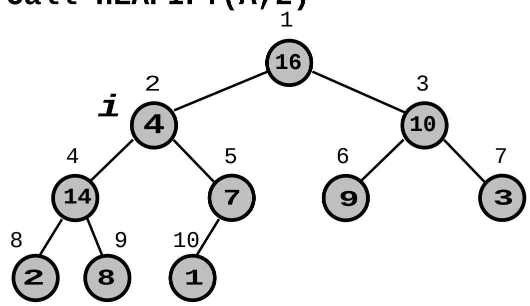
$\triangleright$ Left & Right subtrees of i are heaps.
$\triangleright$ Makes subtree rooted at i a heap.
$l \leftarrow$ LEFT(i) $\triangleright$ $l = 2i$
$r \leftarrow$ RIGHT(i) $\triangleright$ $r = 2i + 1$
if $l \leq n$ and $A[l] > A[i]$
  then $largest \leftarrow l$
  else $largest \leftarrow i$
if $r \leq n$ and $A[r] > A[largest]$
  then $largest \leftarrow r$
if $largest \neq i$
  then exchange $A[i] \leftrightarrow A[largest]$
    HEAPIFY($A, largest$)

Heapify - comments

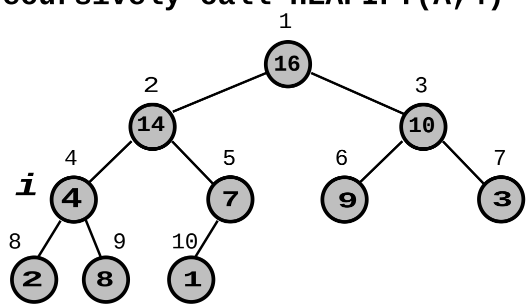
- Correctness: induction on the height of i
- Analysis: time proportional to height of $i = \mathrm{O}(\lg n)$
- For faster code, use a loop instead of recursion.

Heaps: Heapify Example

1. Call HEAPIFY(A,2)

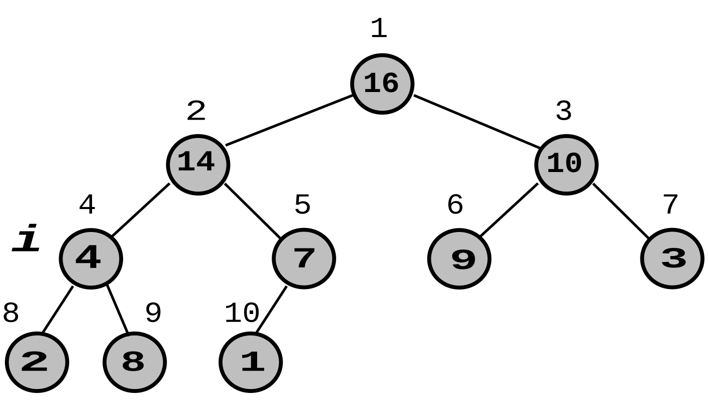


2. Exchange A[2] with A[4] and recursively call HEAPIFY(A,4)



3. Exchange A[4] with A[9] and recursively call HEAPIFY(A,9)

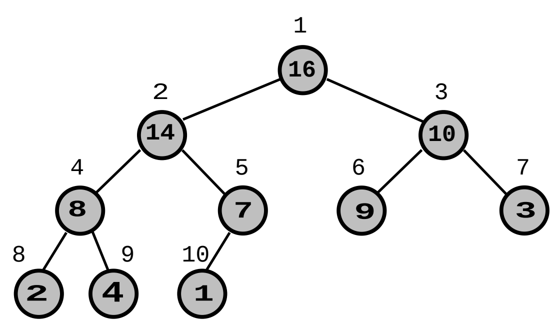
Heaps: Heapify Example (cont)



3. Exchange A[4] with A[9] and recursively call HEAPIFY(A,9)

Heaps: Heapify Example (cont)

4. Node 9 has no children, so we are done.

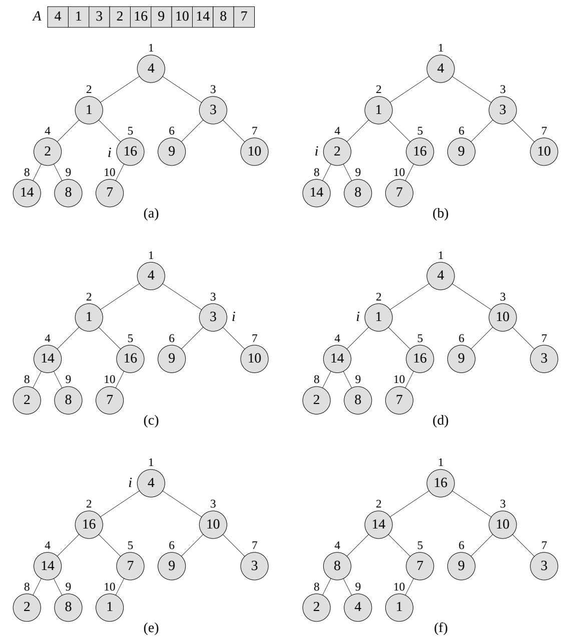


Heaps: Building a heap

Remains to show how to implement INSERT.
But first, we'll do something simpler.

- Convert an array $A[1\,..\,n]$, where $n = \mathit{length}[A]$, into a heap.
- Notice that the elements in the subarray $A[(\lfloor n/2 \rfloor + 1)\,..\,n]$ are already 1-element heaps to begin with.

BUILD-HEAP(A)
  for $i \leftarrow \lfloor n/2 \rfloor$ **downto** 1
    do HEAPIFY(A, i)



Heaps: BUILD-HEAP: Rough Analysis

- Correctness: induction on i, All trees rooted at $m > i$ are heaps.
- Running time: n calls to HEAPIFY $= n$ O($\lg n$) $=$ O($n \lg n$)
- This is good enough for an O($n \lg n$) bound on HEAPSORT, but sometimes we build heaps for other reasons, so....

Heaps: BUILD-HEAP: Tighter Analysis

height of node: longest path from node to leaf
height of tree: height of root

BUILD-HEAP(A)
  for $i \leftarrow \lfloor n/2 \rfloor$ **downto** 1
    do HEAPIFY(A, i)

Time of Heapify = O(height of subtree rooted at i)
Assume $n = 2^k - 1$ (a complete binary tree)

$$\begin{aligned} T(n) &= O\left(\frac{n+1}{2} + \frac{n+1}{4} \cdot 2 + \frac{n+1}{8} \cdot 3 + \ldots + 1 \cdot k\right) \\ &= O\left((n+1) \cdot \sum_{i=1}^{k} \frac{i}{2^i}\right) \\ &= O(n) \\ &\quad \text{since } \sum_{i=1}^{k} \frac{i}{2^i} = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2 \end{aligned}$$

Heaps: BUILD-HEAP: Tighter Analysis

How? Use following "trick":

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x} \text{ if } |x| < 1$$

Differentiate:

$$\sum_{i=1}^{\infty} i \cdot x^{i-1} = \frac{1}{(1-x)^2}$$

Mult by x:

$$\sum_{i=1}^{\infty} i \cdot x^{i} = \frac{x}{(1-x)^2}$$

Plug in $x = \frac{1}{2}$:

$$\sum_{i=1}^{\infty} \frac{i}{2^i} = \frac{\frac{1}{2}}{\frac{1}{4}} = 2$$

Therefore BUILD-HEAP time is O(n).

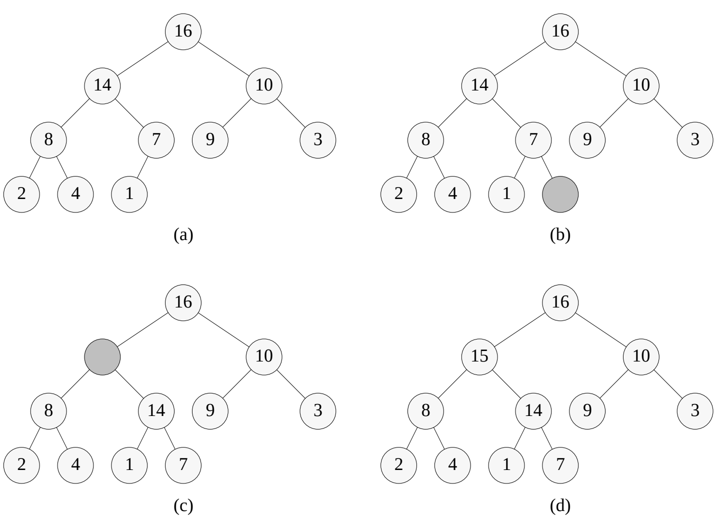
Priority Queue: Insertion

Time for implementation of INSERT.

Main idea: "heapify upwards".

HEAP-INSERT(A, key)
- $heap\text{-}size[A] \leftarrow heap\text{-}size[A] + 1$
- $i \leftarrow heap\text{-}size[A]$
- **while** $i > 1$ and $A[\text{PARENT}(i)] < key$
  - **do** $A[i] \leftarrow A[\text{PARENT}(i)]$
    - $i \leftarrow \text{PARENT}(i)$
- $A[i] \leftarrow key$

Analysis: $O(\lg n)$ time for n element heap.



Heap-Insert

Heaps: Heapsort

HEAPSORT(A)
- BUILD-HEAP(A)
- **for** $i \leftarrow n$ **downto** 2
  - **do** exchange $A[1] \leftrightarrow A[i]$
    - $Heap\text{-}size[A] \leftarrow Heap\text{-}size[A] - 1$
    - HEAPIFY(A, 1)

- total running time: $O(n \lg n)$
- sorts in place !

$\Rightarrow$ best sorting algorithm so far (at least in theory).